# CHƯƠNG V: HÀM

## HÀM

### Khái niệm

Hàm trong C là một tập hợp các xử lý nhằm thực hiện một chức năng cụ thể nào đó trong chương trình. Hàm cho phép bạn kết hợp các xử lý khác nhau thành một và đặt tên cho nó. Sau khi tạo và đặt tên cho một hàm, chúng ta chỉ cần gọi tên hàm ra mỗi khi cần sử dụng đến nó trong chương trình.

Bằng cách kết hợp các xử lý lại thành một hàm, chúng ta không cần viết lại các xử lý nhiều lần, qua đó có thể giảm sai sót khi viết code, cũng như có thể tái sử dụng hàm cho một chương trình khác.

Một hàm có thể nhận một giá trị và trả về một giá trị đã trải qua một số xử lý, do đó nó có khả năng tạo ra công suất tối đa chỉ với lượng mã chương trình tối thiểu.

### Cấu trúc hàm trong C

Khi học về hàm trong C chúng ta cần nắm rõ cấu trúc hàm gồm 3 thành phần chính, đó là **tham số**, **đối số** và **giá trị trả về**.

#### Khai báo hàm trong C

Một khai báo hàm cho trình biên dịch biết về tên hàm và cách gọi hàm. Cơ thể thực tế của hàm có thể được định nghĩa riêng.

**Cú pháp khai báo hàm:**

|  |
| --- |
| < Kiểu dữ liệu trả về > < Tên Hàm > (Tham số truyền vào); |

Trong đó:

* **Kiểu dữ liệu trả về** là kiểu dữ liệu của giá trị trả về từ hàm, ví dụ như int, char. Nếu hàm không trả về giá trị thì chúng ta cần chỉ định kiểu **[void]** để thay thế.
* **Tên Hàm** là một chuỗi ký tự dùng để đặt tên đại diện cho hàm.
* **Tham số** là các biến sử dụng trong khai báo hàm, cũng như để nhận đối số (các giá trị truyền vào) khi chúng ta gọi hàm.

Khai báo hàm không cần thiết nếu hàm do người dùng định nghĩa hàm được xác định trước hàm **main()**

Khai báo hàm là **bắt buộc** khi bạn định nghĩa một hàm trong một tệp nguồn và bạn gọi hàm đó trong một tệp khác. Trong trường hợp này, bạn nên khai báo hàm ở **đầu tệp gọi hàm**

#### Định nghĩa hàm

Chúng ta viết **kiểu dữ liệu trả về** từ hàm trước, sau đó đến **tên hàm**, rồi kiểu và tên của các **tham số truyền vào** sử dụng trong hàm, và cuối cùng là các **xử lý trong hàm** nằm giữa cặp ngoặc nhọn {} cùng với**giá trị trả về**, với cú pháp hàm như sau:

|  |
| --- |
| < Kiểu dữ liệu trả về > < Tên Hàm > (Tham số truyền vào){     Câu lệnh 1 trong hàm;     Câu lệnh 2 trong hàm;     ...     return < Giá trị trả về >;  } |

Trong đó:

* **Kiểu dữ liệu trả về** là kiểu dữ liệu của giá trị trả về từ hàm, ví dụ như int, char. Nếu hàm không trả về giá trị thì chúng ta cần chỉ định kiểu **[void]** để thay thế.
* **Tên Hàm** là một chuỗi ký tự dùng để đặt tên đại diện cho hàm.
* **Tham số** là các biến sử dụng trong khai báo hàm, cũng như để nhận đối số (các giá trị truyền vào) khi chúng ta gọi hàm.
* **return** là từ khóa dùng để trả giá trị trả về từ hàm trong C
* **Giá trị trả về** là các đối tượng được trả về sau khi hàm được thực thi. Ví dụ như hàm có thể trả về một số, một chuỗi, hoặc một con trỏ v.v.

#### Gọi hàm

Trong khi tạo một hàm C, bạn đưa ra một định nghĩa về chức năng của hàm. Để sử dụng một hàm, bạn sẽ phải gọi hàm đó để thực hiện tác vụ được xác định.

Khi một chương trình gọi một hàm, điều khiển chương trình được chuyển đến hàm được gọi. Một hàm được gọi thực hiện một nhiệm vụ đã định nghĩa và khi câu lệnh trả về của nó được thực hiện hoặc khi nó kết thúc bằng hàm đóng, nó sẽ trả về chương trình điều khiển quay trở lại chương trình chính.

Để gọi một hàm, bạn chỉ cần chuyển các tham số bắt buộc cùng với tên hàm và nếu hàm trả về một giá trị, thì bạn có thể lưu trữ giá trị trả về.

|  |
| --- |
| Tên Hàm( đối số 1 , đối số 2 ,...); |

Chúng ta viết tên hàm, sau đó đặt các đối số (các giá trị truyền vào khi gọi hàm) ở giữa cặp dấu ngoặc đơn **()** và cách nhau bởi dấu phẩy **,** . Các giá trị của đối số sẽ được dùng để truyền tham số trong C.

Ví dụ

|  |
| --- |
| #include <stdio.h>    /\* khai bao ham \*/  int max(int *num1*, int *num2*);    int main () {       /\* dinh nghia bien local \*/      int a = 100;      int b = 200;      int ret;       /\* goi mot ham de lay gia tri lon nhat \*/      ret = max(a, b);        printf( "Max value is : %d\n", ret );        return 0;  }    /\* ham tra ve gia tri lon nhat giua hai so \*/  int max(int *num1*, int *num2*) {       /\* dinh nghia bien local \*/      int result;        if (*num1* > *num2*)          result = *num1*;      else          result = *num2*;        return result;  } |

Kết quả

|  |
| --- |
| Max value is : 200 |

### Tham số và đối số trong C

#### Tham số là gì? | Parameter

**Tham số (parameter)** là các biến có kiểu và tên được sử dụng để nhận các giá trị truyền vào (Đối số- argument) để xử lý trong hàm. Tham số được sử dụng khi chúng ta **khai báo hàm trong C**.

Các tham số sử dụng trong hàm đều cần chỉ định **kiểu và tên** của nó. Và **kiểu, tên** của tham số cần phải giống với **kiểu, tên** của loại đối tượng mà nó sẽ nhận khi đối tượng đó được **truyền vào hàm.**

Ví dụ, nếu bạn muốn truyền giá trị là một số thuộc **kiểu int**, thì bắt buộc là tham số khai báo trong hàm cũng phải thuộc **kiểu int.** Tương tự nếu đối tượng truyền vào là một **char, hay một chuỗi,** bạn cũng cần phải chỉ định kiểu **char hay char[]** tương ứng.

**VD:**

Các biến a,b sử dụng trong hàm được gọi là tham số, và chúng ta sử dụng **tham số** để khai báo hàm trong C

|  |
| --- |
| int find\_sum (int a, int b){     int sum = a + b;     return sum;  } |

#### Đối số là gì? | Argument

**Đối số (argument)** là các giá trị thực được truyền vào khi **gọi hàm**. Đối số được truyền vào hàm qua sẽ được gán vào tham số, và được sử dụng khi chúng ta **gọi hàm trong C**.

|  |
| --- |
| int find\_sum (int a, int b){     int sum = a + b;     return sum;  }  int main(){     int tong = find\_sum(1 , 2);  }  Các giá trị thực truyền vào khi ta gọi hàm được gọi là **đối số**, và chúng ta dùng đối số để truyền tham số trong C |

Có nhiều loại giá trị có thể truyền vào hàm khi chúng ta gọi hàm, ví dụ như biến, con trỏ và cấu trúc chẳng hạn. Với mỗi loại đối tượng như vậy sẽ có kiểu dữ liệu khác nhau, và chúng ta cần cần chú ý phải truyền đối tượng vào hàm có **kiểu dữ liệu** **giống** với **kiểu của tham số tương ứng** trong hàm dùng để nhận nó.

Ví dụ nếu khi khai báo hàm, chúng ta sử dụng một tham số có kiểu **int,** thì chúng ta cũng chỉ có thể truyền đối số có cùng kiểu **int** vào hàm mà thôi. Nếu không thì hàm không chạy được và lỗi sẽ bị xảy ra.

#### Sự khác biệt giữa một đối số và một tham số

Tổng kết sự khác nhau giữa **đối số** và **tham số**.

* **Tham số** là các biến có tên được sử dụng trong khai báo hàm để nhận giá trị từ các đối số.
* **Đối số** của hàm là các giá trị thực được truyền vào khi **gọi hàm**.
* Tham số nhận giá trị của các đối số được truyền vào từ **ngoài hàm** và thực hiện tính toán bên **trong hàm.**

### Giá trị trả về trong C

#### Return trong C là gì

**Return trong C** là một từ khóa (keyword) dùng để trả về giá trị từ hàm. Return có tác dụng kết thúc hàm và trả lại điều khiển cũng như kết quả xử lý hàm cho người gọi. Chúng ta có thể sử dụng hoặc lược bỏ return khi khai báo hàm trong C, và một hàm có chứa return trong C được gọi là hàm trả về giá trị trong C.

|  |
| --- |
| int find\_sum (int a, int b){     int sum = a + b;     return sum;  }  **Giá trị trả về** là một từ khóa để trả về giá trị từ hàm. Chúng ta sử dụng return khi khai báo hàm trong C |

#### Cách dùng return trong C

Tùy thuộc vào việc hàm có trả về giá trị hay không mà chúng ta có những cách sử dụng return trong C khác nhau.

##### return trong hàm trả về giá trị

Khi sử dụng return trong hàm trả về giá trị trong C, chúng ta viết giá trị trả về của hàm đằng sau lệnh return như sau:

|  |
| --- |
| <Kiểu dữ liệu trả về> <Tên Hàm> ( kiểu 1 tham số 1, kiểu 2 tham số 2, ...){     Câu lệnh;     Câu lệnh;     return giá-trị-trả-về;  } |

**VD:**

|  |
| --- |
| int find\_sum(int a,int b){     int sum = a + b;     return sum;  }  int main(void){     int ketqua = find\_sum(1,2);     printf("%d",ketqua);     return 0;  } |

**Kết quả:**

|  |
| --- |
| **3** |

##### return trong hàm không trả về giá trị

Trong hàm không trả về giá trị trong C, chúng ta lược bỏ lệnh return, và chỉ định **kiểu dữ liệu trả về** thành kiểu **void** như sau:

|  |
| --- |
| void Tên Hàm( kiểu 1 tham số 1 , kiểu 2 tham số 2, ...){     Câu lệnh;     Câu lệnh;  } |

**VD:**

|  |
| --- |
| void find\_sum(int a,int b){     int sum = a + b;     printf("%d", sum);  }  int main(void){     find\_sum(1,2);     return 0;  } |

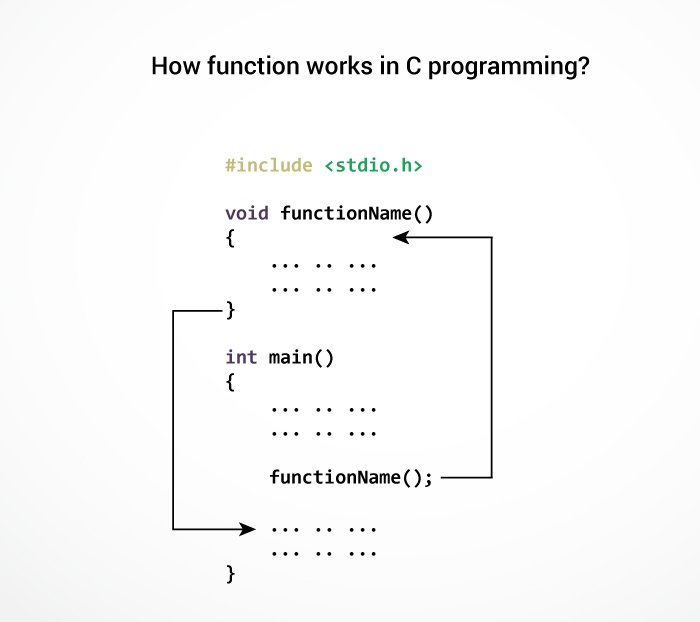
**Kết quả:**

|  |
| --- |
| **3** |

### Cách hoạt động của hàm trong C

Hình ảnh dưới đây cho bạn thấy cách hoạt động của hàm (chương trình con) ở trong ngôn ngữ C. Khi một lời gọi hàm được thực thi thì:

* Chương trình của bạn sẽ nhảy tới nơi định nghĩa hàm đó và thực thi các lệnh từ trên xuống dưới ở trong hàm đó.
* Khi hàm thực hiện xong, chương trình tiếp tục quay về thực hiện các lệnh phía sau lời gọi hàm.

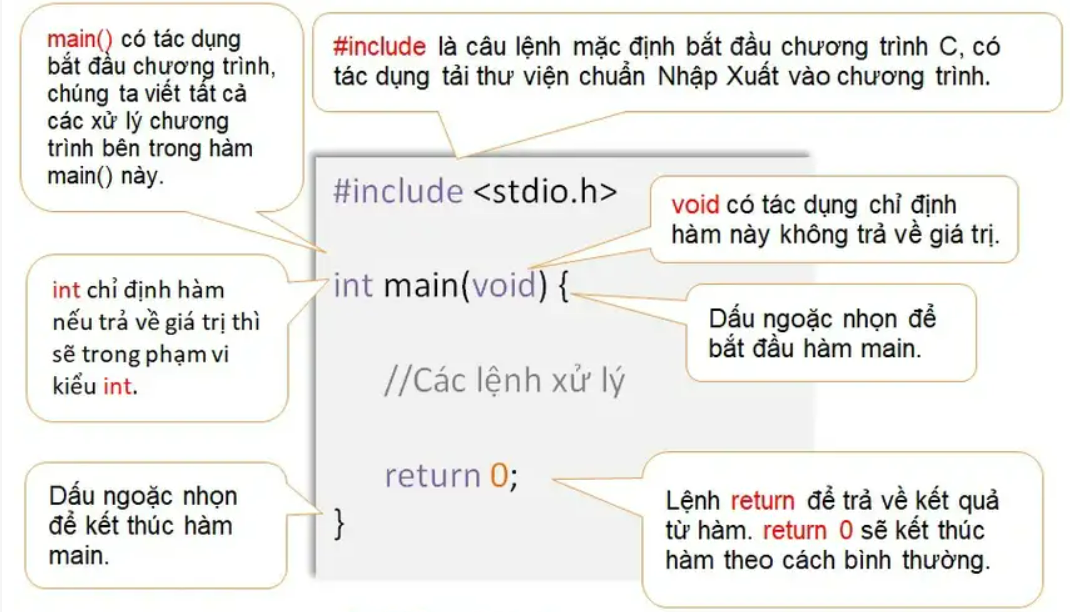


### Phân loại hàm trong C

Sau khi đã biết các định nghĩa sử dụng trong hàm C, sau đây chúng ta sẽ cùng tìm hiểu các loại hàm trong C nhé.  
Trong C có 3 loại hàm, đó là **hàm main trong C**, **các hàm có sẵn trong C**, và **hàm C do người dùng định nghĩa**.

#### Hàm main trong C

Trong ngôn ngữ C, một chương trình là một tập hợp các hàm, với mỗi hàm trong chương trình là “tập hợp các quy trình” cần xử lý. Và trong các hàm đó thì **hàm main() trong C** là **hàm đầu tiên được thực thi** khi bắt đầu chạy một chương trình C.



#### Các hàm có sẵn trong C

**Các hàm có sẵn trong C** là các hàm được chuẩn bị sẵn trong các thư viện chuẩn của C. Các hàm này được tích hợp bên trong các header file, ví dụ như hàm **strlen()** tích hợp trong header file **[string.h**] chẳng hạn.

Các hàm này mặc dù đã được chuẩn bị sẵn, nhưng chúng ta chỉ có thể thực thi chúng trong chương trình C, nếu chúng ta gọi chúng bên trong hàm main() mà thôi.

Lại nữa, để sử dụng được các hàm có sẵn này, chúng ta cần phải thêm (include) các header file này vào đầu mỗi chương trình.

Ví dụ, các hàm như hàm print(), hàm scanf() v.v.. đều là các hàm có sẵn trong tích hợp trong header file **[stdio.h],** hay hàm strlen() lại được tích hợp trong header file **[string.h]**, và chúng ta cần include các file này trong chương trình như sau:

|  |
| --- |
| #include <string.h>  int main(void){     printf("Nhap chuoi ky tu: ");       char str[100];     scanf("%s", str);       int len = strlen(str); //Hàm strlen dùng để tính độ dài chuỗi.     printf("Do dai chuoi: %d",len);     return 0;  } |

**Kết quả:**

|  |
| --- |
| Nhap chuoi ky tu: abc Do dai chuoi: 3 |

#### Hàm do người dùng định nghĩa

**Hàm C do người dùng định nghĩa** (user-defined functions)) là các hàm mà chúng ta tự mình tạo ra bằng cách khai báo chúng trong chương trình. Sau khi khai báo hàm, chúng ta có thể sử dụng chúng trực tiếp trong chương trình bằng cách gọi chúng tại bất kỳ thời điểm nào mà chúng ta muốn.

Hàm C do người dùng định nghĩa có ưu điểm rất lớn, đó là sự tự do điều chỉnh hàm theo ý mà bạn muốn, bởi vì hàm là do bạn tự tạo ra, tự bạn thiết kế và làm chủ nó.

##### Hàm không có tham số truyền vào, không có kiểu dữ liệu trả về

Hàm không có tham số truyền vào và không có kiểu dữ liệu trả về (void) thường được sử dụng để thực hiện các tác vụ không trả về kết quả, như xuất thông tin ra màn hình, đọc dữ liệu từ bàn phím, hoặc thực hiện một tác vụ nào đó nhưng không cần trả về bất kỳ giá trị nào.

Cấu trúc của hàm không có tham số truyền vào và không có kiểu dữ liệu trả về như sau:

|  |
| --- |
| void <Tên Hàm>() {      // khối lệnh của hàm  }  Hoặc  void <Tên Hàm>(void) {  // khối lệnh của hàm  } |

Trong đó:

**void**: là kiểu dữ liệu của hàm không trả về giá trị.

**<Tên Hàm>:** là tên của hàm.

**khối lệnh của hàm:** là tập hợp các lệnh và biểu thức để thực hiện các tác vụ cần thiết.

**Ví dụ**

|  |
| --- |
| #include <stdio.h>  void Nhap\_\_Tinh\_\_In();  /// Hàm con  void  Nhap\_\_Tinh\_\_In() {      int a, b, sum = 0;      printf("Nhap a: ");      scanf("%d", &a);      printf("Nhap b: ");      scanf("%d", &b);      sum = a + b;      printf("Tong: %d", sum);  }    int main() {      Nhap\_\_Tinh\_\_In();  } |

**Kết quả:**

|  |
| --- |
| **Nhap a: 5**  **Nhap b: 4**  **Tong: 9** |

Như bạn thấy, hàm **Nhap\_\_Tinh\_\_In()** không có tham số đầu vào, bản thân nó tự thực hiện nhận giá trị từ bàn phím, kiểm tra và sau đó cũng in ra kết quả luôn. Vì là nó không trả về giá trị nên chúng ta dùng kiểu void

##### Hàm không có tham số truyền vào, có kiểu dữ liệu trả về

Hàm không có tham số truyền vào, có kiểu dữ liệu trả về là một loại hàm trong ngôn ngữ lập trình C. Hàm này không yêu cầu bất kỳ tham số đầu vào nào, nhưng sẽ trả về một giá trị theo kiểu dữ liệu được xác định trước.

Cấu trúc của hàm không có tham số truyền vào, có kiểu dữ liệu trả về này như sau:

|  |
| --- |
| < Kiểu dữ liệu trả về > < Tên Hàm >() {      // Khối lệnh của hàm     return < giá trị trả về >;  }  **Hoặc**  < Kiểu dữ liệu trả về > < Tên Hàm >(void) {  // Khối lệnh của hàm  return < giá trị trả về >;  } |

Trong đó:

**< Kiểu dữ liệu trả về >:** là kiểu dữ liệu của giá trị trả về của hàm.

**< Tên Hàm >:**  là tên của hàm.

**Khối lệnh của hàm** là tập hợp các lệnh thực hiện các công việc cần thiết.

**return** được sử dụng để trả về một giá trị có kiểu dữ liệu được xác định trước từ hàm. Giá trị này sẽ được truyền cho lời gọi hàm.

Ví dụ:

|  |
| --- |
| #include <stdio.h>  void Nhap\_\_Tinh\_\_In();  /// Hàm con  int Nhap\_\_Tinh() {      int a, b, sum = 0;      printf("Nhap a: ");      scanf("%d", &a);      printf("Nhap b: ");      scanf("%d", &b);      sum = a + b;        return sum;  }    int main() {        int kq = Nhap\_\_Tinh();        printf("Tong: %d", kq);  } |

**Kết quả:**

|  |
| --- |
| **Nhap a: 5**  **Nhap b: 4**  **Tong: 9** |

##### Hàm có tham số truyền vào, không có kiểu dữ liệu trả về

Hàm có tham số truyền vào, không có kiểu dữ liệu trả về là loại hàm trong ngôn ngữ lập trình C mà nhận vào một hoặc nhiều tham số, thực hiện một số tác vụ và không trả về bất kỳ giá trị nào.

Cấu trúc của một hàm có tham số truyền vào, không có kiểu dữ liệu trả về như sau:

|  |
| --- |
| void <Tên Hàm> ( kiểu 1 tham số 1 , kiểu 2 tham số 2, ...){     // khối lệnh của hàm  } |

Trong đó

**void**: là kiểu dữ liệu của hàm không trả về giá trị.

**<Tên Hàm>:** là tên của hàm.

**kiểu:** là kiểu dữ liệu của tham số truyền vào. Bạn có thể có nhiều tham số hơn nếu cần thiết, mỗi tham số phân cách nhau bằng dấu phẩy.

**tham số:** là một biến được định nghĩa trong phần khai báo của hàm, nhằm nhận giá trị truyền vào cho hàm khi được gọi

**khối lệnh của hàm:** là tập hợp các lệnh và biểu thức để thực hiện các tác vụ cần thiết.

**Ví dụ**

|  |
| --- |
| #include <stdio.h>  void Tinh\_\_In();  /// Hàm con  void Tinh\_\_In (int *x1*, int *x2*) {      int sum = *x1* + *x2*;        printf("Tong: %d", sum);  }    int main() {        int a, b, sum = 0;      printf("Nhap a: ");      scanf("%d", &a);      printf("Nhap b: ");      scanf("%d", &b);        Tinh\_\_In(a, b);  } |

**Kết quả:**

|  |
| --- |
| **Nhap a: 5**  **Nhap b: 4**  **Tong: 9** |

##### Hàm có tham số truyền vào, có kiểu dữ liệu trả về

Hàm có tham số truyền vào và có kiểu dữ liệu trả về là loại hàm cho phép chúng ta tính toán hoặc xử lý dữ liệu dựa trên giá trị được truyền vào qua các tham số và trả về kết quả sau khi hoàn thành xử lý.

Cấu trúc của hàm có tham số truyền vào có kiểu dữ liệu trả về bao gồm:

|  |
| --- |
| < Kiểu dữ liệu trả về > < Tên Hàm >(kiểu 1 tham số 1 , kiểu 2 tham số 2, ...) {      // Khối lệnh của hàm     return <giá trị trả về>;  } |

Trong đó

**< Kiểu dữ liệu trả về >:** là kiểu dữ liệu của giá trị trả về của hàm.

**<Tên Hàm>:** là tên của hàm.

**kiểu:** là kiểu dữ liệu của tham số truyền vào. Bạn có thể có nhiều tham số hơn nếu cần thiết, mỗi tham số phân cách nhau bằng dấu phẩy.

**tham số:** là một biến được định nghĩa trong phần khai báo của hàm, nhằm nhận giá trị truyền vào cho hàm khi được gọi

**khối lệnh của hàm:** là tập hợp các lệnh và biểu thức để thực hiện các tác vụ cần thiết.

Ví dụ về hàm có tham số truyền vào không có dữ liệu trả về để tính tổng 2 số a và b

**return** được sử dụng để trả về một giá trị có kiểu dữ liệu được xác định trước từ hàm. Giá trị này sẽ được truyền cho lời gọi hàm.

**Ví dụ**

|  |
| --- |
| #include <stdio.h>  // Khai báo hàm Tinh  int Tinh(int *x1*, int *x2*);  // Định nghĩa hàm Tinh  int Tinh(int *x1*, int *x2*) {      int sum = *x1* + *x2*;      return sum;  }  int main() {      int a, b, sum = 0;      printf("Nhap a: ");      scanf("%d", &a);      printf("Nhap b: ");      scanf("%d", &b);      sum = Tinh(a, b);      printf("Tong: %d\n", sum);      return 0;  } |

**Kết quả:**

|  |
| --- |
| **Nhap a: 5**  **Nhap b: 4**  **Tong: 9** |

## TRUYỀN THAM CHIẾU VÀ TRUYỀN THAM TRỊ

Một điều khá quan trọng trong C đó là **truyền tham chiếu và tham trị vào một phương thức (Function)**.

### Truyền tham chiếu

Truyền tham chiếu chính là cách chúng ta truyền cho nó một bản gốc thông qua địa chỉ ‘**&**‘. Nghĩa là giá trị của biến tham số truyền vào hàm, sẽ là giá trị của biến truyền vao [**hàm**](https://khuenguyencreator.com/ham-la-gi-cach-su-dung-ham-trong-lap-trinh-c/) đó.

**Nói đơn giản truyền tham chiếu là truyền địa chỉ.**

**Xét ví dụ:**

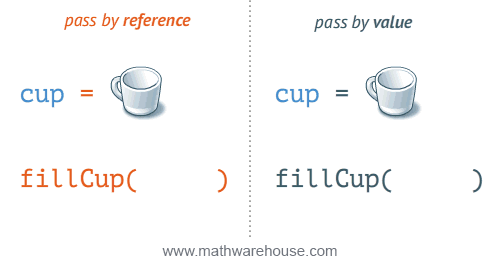
|  |
| --- |
| #include <stdio.h>  void giam1donvi (int \**var*){      \**var* = \**var* -1 ;  }  int main() {      int x = 100;      printf("Gia tri ban dau: %d\n", x);      giam1donvi(&x);      printf("Gia tri sau khi da giam: %d", x);      return 0;  } |

**Kết quả:**

|  |
| --- |
| Gia tri ban dau: 100  Gia tri sau khi da giam: 99 |

**Giải thích:**

Ta thấy rằng ở hàm **giam1donvi** chúng ta truyền vào địa chỉ của biến x (&x). Trong hàm này đã tác động trực tiếp vào vùng ô nhớ cấp phát để lưu giá trị của biến x trên RAM. Do đó sau khi ra khỏi hàm thì giá trị của biến x đã bị thay đổi.



*Ví dụ về truyền tham chiếu (pass by reference) và truyền tham trị (pass by value)*

### Truyền tham trị

Truyền tham trị là truyền cho đối số một bản sao, nghĩa là giá trị của biến đó sẽ được sao chép sang biến tham số truyền vào hàm. Đây cũng là cách chúng ta sử dụng hàm bình thường.

**Nói đơn giản truyền tham trị là truyền giá trị.**

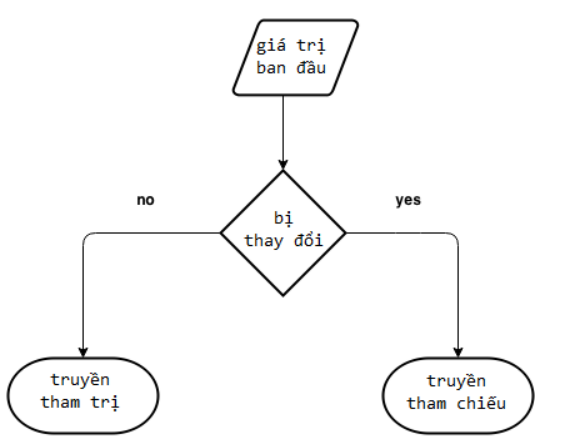
**Xét ví dụ:**

|  |
| --- |
| #include <stdio.h>  void incValue(int x)  {      x++;      printf("Bien trong ham = %d \n", x);  }  int main ()  {      int a = 10;      incValue(a);      printf("Bien ngoai ham ham = %d \n", a);      return 0;  } |

**Kết quả:**

|  |
| --- |
| **Bien trong ham = 11**  **Bien ngoai ham ham = 11** |

Khi truyền vào biến a cho hàm incValue, thực chất trình biên dịch sẽ copy giá trị của a cho x ( x = a). Sau đó lấy giá trị đó để tính toán. Thế nên, khi chúng ta thay đổi giá trị của x, giá trị của a không thay đổi



## HÀM ĐỆ QUY

Đệ quy trong C là quá trình trong đó một phương thức gọi lại chính nó một cách liên tiếp. Một phương thức trong C gọi lại chính nó được gọi là phương thức đệ quy.

Sử dụng đệ quy giúp code chặt chẽ hơn nhưng sẽ khó để hiểu hơn.

Cú pháp:

|  |
| --- |
| < Kiểu dữ liệu trả về > < tenhamdequi > () {     // khối lệnh của hàm     tenhamdequi(); /\* gọi lại chính nó \*/  }  int main() {     tenhamdequi();  } |

Do tính chất tự gọi lại chính nó nên để tránh việc hàm đệ quy chạy không bao giờ dừng bạn luôn cần có điểm dừng (điểm dừng được hiểu đơn giản là tới 1 lúc nào đó, hàm đệ quy sẽ không gọi lại chính nó nữa).

**VD**: Cho số nguyên n được nhập từ bàn phím, bạn hãy viết hàm đệ quy trả về n! (n giai thừa).

|  |
| --- |
| #include <stdio.h>  int factorial(int n) {      if (n == 1) return 1;      return n\* factorial(n-1);  }  int main() {      int n;  printf("Nhap vao gia tri n :");      scanf("%d", &n);      printf("%d", factorial(n));      return 0;  } |

**Kết quả:**

|  |
| --- |
| **Nhap vao gia tri n :5**  **120** |

Mô tả hàm đệ quy tính 5!:

|  |
| --- |
| factorial(5)  = 5 \* factorial(4)  = 5 \* 4 \* factorial(3)  = 5 \* 4 \* 3 \* factorial(2)  = 5 \* 4 \* 3 \* 2 \* factorial(1)  = 5 \* 4 \* 3 \* 2 \* 1  = 120 |

Điểm dừng ở ví dụ trên chính là khi đầu vào của hàm factorial bằng 1 thì hàm factorial sẽ trả về 1 thay vì gọi tiếp tới chính nó.

**VD:** Cho số nguyên n được nhập từ bàn phím, bạn hãy viết hàm đệ quy trả về tổng các số từ 1 tới n.

|  |
| --- |
| #include <stdio.h>  int sum(int n) {      if (n == 1) return 1;      return n + sum(n-1);  }  int main() {      int n;      printf("Xin moi nhap gia tri n: ");      scanf("%d", &n);      printf("%d", sum(n));      return 0;  } |

**Kết quả:**

|  |
| --- |
| **Xin moi nhap gia tri n: 5**  **15** |